On verifying and maintaining connectivity of interval temporal networks

Eleni C. Akrida[1] and Paul G. Spirakis[1,2]

[1] Department of Computer Science, University of Liverpool, UK
[2] Computer Engineering & Informatics Department, University of Patras, Greece
{Eleni.Akrida,P.Spirakis}@liverpool.ac.uk

Abstract. An interval temporal network is, informally speaking, a network whose links change with time. The term *interval* means that a link may exist for one or more time intervals, called *availability intervals* of the link, after which it does not exist (until, maybe, a further moment in time when it starts being available again). In this model, we consider continuous time and high-speed (instantaneous) information dissemination. An interval temporal network is connected during a period of time $[x, y]$, if it is connected for all time instances $t \in [x, y]$ (instantaneous connectivity). In this work, we study instantaneous connectivity issues of interval temporal networks. We provide a polynomial-time algorithm that answers if a given interval temporal network is connected during a time period. If the network is not connected throughout the given time period, then we also give a polynomial-time algorithm that returns large components of the network that remain connected and remain large during $[x, y]$; the algorithm also considers the components of the network that start as large at time $t = x$ but dis-connect into small components within the time interval $[x, y]$, and answers how long after time $t = x$ these components stay connected and large. Finally, we examine a case of interval temporal networks on tree graphs where the *lifetimes* of links and, thus, the failures in the connectivity of the network are not controlled by us; however, we can "feed" the network with extra edges that may re-connect it into a tree when a failure happens, so that its connectivity is maintained during a time period. We show that we can with high probability maintain the connectivity of the network for a long time period by making these extra edges available for re-connection using a randomized approach. Our approach also saves some cost in the design of availabilities of the edges; here, the cost is the sum, over all extra edges, of the length of their availability-to-reconnect interval.

1 Introduction and motivation

A great variety of systems in society, technology and nature can be modeled as networks, linked with edges; from the Internet to the web of social acquaintances, from the transport network of a city to the nervous system of the human body. The structure of a network describes the several connections between the participating entities and helps us understand or predict the behavior of dynamical

systems. However, in many cases the links between the participating entities do not always remain active but change or disappear as time progresses. A *temporal network* is, informally speaking, a network that changes with time. Both traditional and modern networks, such as communication networks, social networks, transportation networks and physical systems, can be modeled as temporal.

Dynamic networks in general have been attracting attention over the past years, exactly because they model real-life applications. The study of temporal networks in particular is quite interdisciplinary, which is also reflected in literature where the object of study may have different names - temporal graphs, temporal networks, time-varying graphs, evolving graphs, time-stamped graphs etc. Kempe et al. [14] considered the single-labeled discrete-time model of temporal graphs, where every edge may become available (for use) only at a discrete moment in time, called the *label* of the edge; their main motivation was to examine how basic graph properties change in this temporal setting. In their multi-labeled model, Mertzios et al. [17] extended the model of [14] to many labels per edge and mainly examined the number of labels needed for a temporal design of a network to guarantee several graph properties with certainty. They also provided an algorithm to compute foremost time-respecting paths; in this discrete-time model, a time-respecting path is a path in which successive edges have strictly increasing time labels and a foremost time-respecting path is one that *reaches* the destination vertex at the earliest possible time. Random edge availabilities in the discrete-time model of temporal networks were first considered by Akrida et al. [1] in order to study the Expected Temporal Diameter of temporal graphs.

Assuming the availability of an edge for a whole time-interval $[t_1, t_2]$ or multiple such time-intervals, and not just for discrete moments, is a clearly natural assumption since time is indeed a continuous measure. Bui-Xuan et al. [4] consider a class of dynamic networks where the changes in the topology can be predicted in advance and in which each node and each edge comes with a list of time intervals; they give algorithms for computing foremost time-respecting paths, shortest (minimum hop count) time-respecting paths and fastest (minimum time) time-respecting paths in this model. Fleischer and Tardos [11] consider a continuous-time model of dynamic graphs and prove continuous versions of known discrete-time flow algorithms for dynamic flow problems. Fleischer and Skutella [10] and Koch et al. [15] also engage in the study of flows in a continuous-time model of dynamic graphs.

Further relevant studies on networks labeled by time units or segments, in addition to the ones mentioned above, include:

- *Labeled Graphs:* Labeled graphs have been widely used both in Computer Science and in Mathematics [19].
- *Dynamic Distributed Networks:* In recent years, there is a growing interest in distributed computing systems that are inherently dynamic [2, 3, 5, 6, 9, 16, 18, 20, 21].

– *Distance labeling:* A distance labeling of a graph G is an assignment of unique labels to vertices of G so that the distance between any two vertices can be inferred from their labels alone [12, 13].

1.1 Our contribution

In this work, we restrict our attention to *continuous time* and consider systems in which only the connections between the participating entities may change, while the entities themselves remain unchanged. So we consider networks of a fixed vertex set, each edge e of which is available over a set of time intervals $L_e = \{[t_1, t'_1], \ldots, [t_k, t'_k]\}$. Each interval indicates a period of availability of e; the unprimed times mark the start of the availability period and the primed times mark the end. This is a model that could naturally represent several systems, such as proximity networks where a link may represent that two entities have been close to each other for some extent of time, or infrastructural systems like the Internet, or even seasonal food webs where a time interval may represent the fact that one species is the main food source of another for a specific period of the year.

We give a polynomial-time deterministic algorithm that decides if a given interval temporal network is connected during a given period (cf. Section 3); if the network is not connected, the algorithm returns the maximal interval from the beginning of the given period during which the network stays connected. We then provide a polynomial-time algorithm that decides if a given interval temporal network has *large enough* connected components during a given time period; here, the size of the components in question is determined by a parameter provided by the user as input to the algorithm (cf. Section 4). Finally, we provide a probabilistic analysis of a scenario where the *lifetime* of the intervals assigned to the edges of a network on a tree graph are not designed via a deterministic process and are unknown to us; instead, the edges may fail unexpectedly and we are required to supply the network with more available edges so that, when a break in the connectivity of the network happens, we can re-connect it. We wish not to keep all these extra edges available for re-connection at all times, i.e., we wish to maintain connectivity but by paying a low cost on keeping extra edges available. Assuming that the cost of keeping additional edges available is linear to the sum of lengths of their availability intervals, we show a low cost construction. Other work for maintaining some structure or property like connectivity in probabilistic dynamic graphs includes [7, 8].

2 Preliminaries

We focus here on networks, the links of which are not always available. The availability of a link is described via a set of time intervals, one set L_e per edge (arc) e.

Definition 1 (Interval temporal network). *Let $G = (V, E)$ be a (di)graph. An interval temporal network on G is an ordered triplet $G(L) = (V, E, L)$,*

where $L = \{L_e = \{[t_1, t'_1], \ldots, [t_{k_e}, t'_{k_e}]\},$ *for some* $k_e \in \mathbb{N},\ t_i, t'_i \in \mathbb{R}^+,\ i = 1, 2, \ldots,\ k_e : e \in E\}$ *is an* assignment *of availability intervals to the edges (arcs) of* G. L *is called a* labeling *of* G.

The availability intervals of an edge (arc) e represent the *continuous time intervals* at which e is active. When we say that an edge (arc) is active or available during the interval $[a, b]$, for some $a, b \in \mathbb{N}$, it means that the edge exists in the network $\forall t \in \mathbb{R}^+,\ t \in [a, b]$. For the analysis throughout the paper, we assume the intervals $[t_1, t'_1], \ldots, [t_{k_e}, t'_{k_e}]$ to be disjoint.[3] Every time a change in the network happens, i.e., an edge starts or stops being available, we have changes in the topology of the network; so, in a sense, an interval temporal network can be viewed as a sequence of graphs, one after every topology change. However, representing such networks as evolving graphs, i.e., the sequence of states of the network after each change, is not as efficient. The interval representation is indeed a very compact representation of such kinds of evolving graphs.

A basic assumption that we follow is that when a message or an entity passes through an available link at time t, then it can pass through a subsequent link only at some time $t' \geq t$ and only at a time at which that link is available. However, unlike what is assumed in the discrete-time model of [1, 14, 17], here we consider instant information dissemination through a path of the underlying (di)graph, if the consecutive edges (arcs) are consistently labeled In fact, our model considers very high speed of information dissemination, resembling fiber-optic communication, but the small time needed to send a message through a link is considered negligible for the analysis. Consider, for example, the interval temporal network of Figure 1, where the availability intervals of each edge are drawn next to the edge; here, information may start at time $t = 2.5$ from vertex u and arrive at time $t = 2.5$ at vertex z, since all three consecutive edges are available at time $t = 2.5$ (their availability intervals are overlapping).

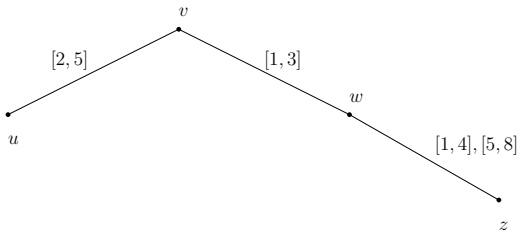

Fig. 1. A model of "Fiber-optic"-like communication.

Definition 2 (Connectivity of interval temporal networks). *An interval temporal network* $G(L) = (V, E, L)$ *is connected at a given time instance* t_0 *if*

[3] We can assume this, because if an edge $e \in E$ has overlapping availability intervals, then we can consider their union as an availability interval of e.

the edges that are available at time t_0, i.e., the edges that have an availability interval which includes t_0, induce a spanning tree.

3 Connectivity of interval temporal networks during a given time period

A fundamental issue for any given network, dynamic or not, is to verify if the network is connected (over time, in the dynamic case), i.e., information can travel via edges between any ordered pair of vertices in it. In this section, we consider interval temporal networks and address the issue of their connectivity.

One can think of an interval temporal network as a dynamic network, where the changes in the topology of the network happen whenever an availability interval of an edge starts or finishes, but can view it as static in between these (instantaneous) changes. Since information can travel instantaneously in interval temporal networks, for such a network to be connected over a time period, all the instances of the "static" networks that are formed during that period need to be connected.

We provide below a polynomial-time procedure to determine if a given interval temporal network is connected throughout a particular time period. Henceforth, we denote by $E(t)$ the set of edges that are available at time t, and t is *not* the finish time of the availability interval that includes t.[4]

Theorem 1. *There is a polynomial-time algorithm (cf. Algorithm 1) which, given an interval temporal network $G(L)$ on n vertices and numbers $x, y \in \mathbb{R}^+$, $x < y$, answers whether $G(L)$ is connected during the time period $[x, y]$, i.e., is connected for every time instance $t \in [x, y]$. If for some $a \in [x, y]$, $[x, a] \subseteq [x, y]$ is the maximal sub-period of $[x, y]$ during which $G(L)$ remains connected, then the algorithm also returns the length of that period, $a - x$.*

Description of the algorithm. The idea behind Algorithm 1 is that $G(L)$ is connected during a period $[x, y]$ if and only if $G(L)$ has a spanning tree for every time instance in $[x, y]$.

Initially, Algorithm 1 finds a spanning tree of the input network $G(L)$ at time x. If no such tree exists, then at time x the network is disconnected and the algorithm terminates. If a spanning tree T exists at time x, then T remains connected until one (or more) of its edges stop being available. Denote by b_1 the first moment in time at which T disconnects. T consists now of a number of connected components and, in fact, T is a forest (collection of trees). The algorithm checks whether there are edges of $G(L)$ that are available at time b_1, which can be added to T and re-connect it. More specifically, the algorithm finds an edge that is available at time b_1 and has endpoints in different connected components of T. The algorithm adds that edge to T and checks if this addition

[4] $E(t)$ are the edges that are available at t and do not stop being available (immediately) after time t.

Algorithm 1 Connectivity of interval temporal networks

Input: A temporal network $G(L)$ of n vertices and numbers $x, y \in \mathbb{R}^+$ such that $x < y$
Output: Answer if $G(L)$ remains connected during the time interval $[x, y]$

1: **if** $E(x)$ induces a spanning tree, T, of G **then**
2: Sort the edges in T according to the finish time of their availability interval; // For every edge in T, we only consider the interval that includes x
3: Let $A = \{e_i,$ with interval $[a_i, b_i] : i = 1, \ldots, n-1\}$ be the sorted list;
4: **if** $b_1 \geq y$ **then**
5: **return** "Network is connected" **and** "Duration of survival =" $y - x$; // If all edges in T remain available until (at least) time y
6: **else**
7: $E' := \{e \in E(T) : b_e = b_1\}$; // b_1 is the first time instance at which T becomes disconnected. E' is the set of edges of T that stop being available at time b_1.
8: $T := T \setminus E'$; // T is now a forest, i.e., consist of a collection of trees
9: Remove E' from A;
10: Let $T_1, T_2, \ldots, T_i, i \in \mathbb{N}$ be the connected components of T;
11: **while** T is disconnected **do**
12: **if** $\exists j, k = 1, 2, \ldots, i : \exists e = (u, v) \in E(b_1) : u \in V(T_j) \wedge v \in V(T_k)$ **then**
13: Find the T_j, T_k trees of T that e connects; // If there is an edge of G with endpoints in different connected components of T and is available at time b_1, then add it to T
14: Merge T_j, T_k and e into a single tree;
15: Update the number i of connected components of T;
16: Insert e in the sorted list A;
17: **else**
18: **return** "Network is disconnected" **and** "Duration of survival =" $b_1 - x$;
19: Break;
20: Go to line 4
21: **else**
22: **return** "Network is disconnected" **and** "Duration of survival =" $t - x$;

re-connects it. If not, then it looks for yet another edge that is available at time b_1 and has endpoints in different connected components of (the current) T. This process continues until T is re-connected or we cannot find any more edges of $G(L)$ that are available at time b_1 and have endpoints in different connected components of T. If at any step of the process there do not exist edges that can re-connect T, then the algorithm returns that the network is disconnected. However, if we can find appropriate edges to re-connect T, then we form another spanning tree of the network, available from time b_1 onwards, and the same procedure continues. The algorithm answers that the network is connected if we form a spanning tree, all the edges of which are available until the end of the period in question, namely until time y.

Note that Algorithm 1 not only answers whether the given network is connected for all times $t \in [x, y]$, but also gives the duration of connectivity of the network (survival duration) within the interval $[x, y]$.

Correctness of the algorithm. Clearly, by definition of connectivity in the model of interval temporal networks, $G(L)$ is connected during a period $[x, y]$ if and only if $G(L)$ has a spanning tree for every time instance in $[x, y]$. We will now show that if $G(L)$ is connected during $[x, y]$, then Algorithm 1 answers that the network is connected, and if $G(L)$ is not connected during $[x, y]$, then Algorithm 1 answers that the network is disconnected.

If $G(L)$ is disconnected at some point $t_0 \in [x, y]$, then at time t_0, the set of available edges $E(t_0)$ will induce no spanning tree. Algorithm 1 will then look for edges to reconnect the connected components of T but will fail to do so. Therefore, it will answer that the network is disconnected and return duration of survival equal to $t_0 - x$.

Now, suppose that the given $G(L)$ remains connected in $[x, y]$. Let T be the spanning tree that the algorithm considers at time x. If all the edges of T remain available until time y, then the algorithm will answer that the network is connected with duration of survival equal to $y - x$. Suppose now that there are edges in T which stop existing at some moment in $[x, y]$ and let t_0 be the first such moment. The algorithm removes from T all the edges that stop existing at time t_0. T is now disconnected but still contains all the vertices of G. Let G_{t_0} be the subgraph of G induced by $E(t_0)$.[5] Clearly, G_{t_0} is connected since $G(L)$ is connected during $[x, y]$. Also, at time t_0, T is a subgraph of G_{t_0}. We will show by contradiction that there exist edges in G_{t_0} with endpoints in different connected components of T, which can be added to T to re-connect it. Indeed, suppose we cannot do that, i.e., we have added as many edges as possible, if any, to T and we can find no more edges with endpoints in different connected components of (the current) T. Let $T_1, T_2, \ldots, T_i$, for some $i \in \mathbb{N}$ be the connected components of T at that point. Let w, z be two vertices of G which belong in T_j, T_k, $j, k \in \{1, \ldots, i\}$, $j \neq k$, respectively. Since G_{t_0} is connected, there exists a path p from w to z in G_{t_0}. Let w' be the last vertex in p which belongs to T_j and let w'' the vertex following w' in p. Since w' and w'' are in different connected components of T and $\{w', w''\} \in E(G_{t_0})$, there is an edge, namely $\{w', w''\}$, with endpoints in different connected components of T, which we can add to T. This contradicts our assumption. Therefore, we can keep adding edges, as the algorithm indicates, until we re-connect T.

The above process goes on until time $t = y$, or until we have a spanning tree that remains intact until time $t = y$. Then, Algorithm 1 answers that the network is connected and returns duration of survival equal to $y - x$.

Running time. The running time of Algorithm 1 depends on the number of times that the spanning tree changes during $[x, y]$. The spanning tree can only change when one or more edges stop being available, so the above number is in general upper bounded by the total number of intervals assigned to the edges of the network:

$$M = \sum_{e \in E} |L_e|$$

[5] Recall that $E(t_0)$ is the set of all available edges of $G(L)$ at time t_0.

Initially, to find $E(x)$ we need to look at every edge $e \in E$ and decide if x is between the start and finish time of one of e's availability intervals. Performing a binary search on the ordered set of start times *and* the ordered set of finish times of e's availability intervals, we can decide if $e \in E(x)$ in time $O(\log |L_e|)$. So, to compute $E(x)$ and check if it induces a spanning tree, we need time:

$$M' = O(\sum_{e \in E} \log |L_e|)$$

Next, time $O(n \log n)$ is required to sort the edges in T, where n is the number of vertices in the network. Then, for every time T changes, we need time M' to find the new set of available edges at the time. We need time $O(n)$ to find the connected components that can be re-connected by the addition of an available edge at the time and update T. Since we add at most $O(n)$ edges to re-connect T, the addition of all edges and the updates of T take a total of $O(n^2)$ time. Also, time $O(n)$ is required to insert the added edges in the sorted list A. Therefore, the running time of Algorithm 1 is $O\big(M'+n \log n+M \cdot (M'+n^2)\big) = O\big(M \cdot (M'+n^2)\big)$.

4 Large connected components during a given time period

In this section, we examine if, given an interval temporal network $G(L)$ of n vertices, numbers $x, y \in \mathbb{R}^+$ and a parameter $0 \leq \varepsilon \leq 1$, we can find one or more *large enough* subsets of the vertices of G which remain connected and remain large within the time interval $[x, y]$. The matter of how large we want these components to be can be handled by adjusting the parameter ε, which gives us a lower bound of $\varepsilon \cdot n$ on the size of the components we are looking for. In this section, we provide an algorithm that efficiently solves the above problem. Henceforth, a "large enough" connected component will be a component of size at least $\varepsilon \cdot n$.

Notice that any connected component C of $G(L)$, at time $t = x$, that is not large enough can be omitted by any algorithm that solves the above problem. Even if the vertices of C connect with more vertices in $G(L)$ at a later moment in time within $[x, y]$, resulting in a large enough connected component C' of $G(L)$ at that time, C' is not a component that was connected throughout $[x, y]$.

Theorem 2. *There is a polynomial-time algorithm which, given an interval temporal network $G(L)$ on n vertices and numbers $x, y \in \mathbb{R}^+$, $x < y$, returns all subgraphs of G of size $\varepsilon \cdot n$, $0 \leq \varepsilon \leq 1$, that remain connected and large (i.e., is always of size at least $\varepsilon \cdot n$) during the time period $[x, y]$. If $[x, a] \subseteq [x, y]$, $a \in [x, y]$, is the maximal sub-period of $[x, y]$ during which such a component remains connected, then the algorithm also returns the length of that period, $a - x$.*

Algorithm 2 Connectivity of interval temporal graphs

Input: A temporal network $G(L)$ of n vertices, numbers $x, y \in \mathbb{R}^+$ such that $x < y$ and parameter $\varepsilon:\ 0 \leq \varepsilon \leq 1$

Output: All components of $G(L)$ of size $\varepsilon \cdot n$ that remain connected during the time interval $[x, y]$

1: Find the set $E(x)$ of available edges at time x, distinguish the connected components and delete those of size smaller than $\varepsilon \cdot n$;
2: **for** each of the remaining connected components **do**
3: Find a spanning tree, T;
4: $n' = |V(T)|$;
5: Sort the edges in T according to the finish time of their availability interval; // For every edge in T, we only consider the interval that includes x
6: Let $A = \{e_i,$ with interval $[a_i, b_i]:\ i = 1, \ldots, n-1\}$ be the sorted list;
7: **if** $b_1 \geq y$ **then**
8: **return** V(T) **and** "Duration of survival of component = " $y - x$;
9: **else**
10: $E' := \{e \in E(T) : b_e = b_1\}$; // b_1 is the first time instance at which T becomes disconnected. E' is the set of edges of T that stop being available at time b_1.
11: $T := T \setminus E'$;
12: Remove E' from A;
13: Let $T_1, T_2, \ldots, T_i, i \in \mathbb{N}$ be the connected components of T;
14: **while** T is disconnected **and** $|V(T)| = n'$ **do**
15: **if** $\exists j, k = 1, 2, \ldots, i : \exists e = (u, v) \in E(b_1) : u \in V(T_j) \wedge v \in V(T_k)$ **then**
16: Find the T_j, T_k trees of T that e connects; // If there is an edge of G with endpoints in different connected components of T and is available at time b_1, then add it to T
17: Merge T_j, T_k and e into a single tree;
18: Update the number i of connected components of T;
19: Insert e in the sorted list A;
20: **else**
21: **for** each connected component C of T with size smaller than $\varepsilon \cdot n$ **do**
22: $T = T \setminus C$;
23: **return** "Duration of survival of component = " $b_1 - x$;
24: **for** each connected component, C', of T **do**
25: $T := C'$;
26: $n' = |V(T)|$;
27: Go to line 7;

Description of the algorithm Algorithm 2 receives as input an interval temporal network of n vertices and an interval $[x, y]$ during which we want to check whether one or more large components of the network remain connected. The algorithm also takes a non-negative parameter ε no larger than 1. This parameter defines how large we want our components to be; more specifically, the algorithm will only look for components of size (number of vertices) at least $\varepsilon \cdot n$. The algorithm returns all those subsets of the vertices of the initial graph, if any, that remain connected (and large) during $[x, y]$. Furthermore, it returns the

duration of connectivity (*survival duration*) of any large enough component that was connected at time $t = x$ but disconnects at some point in $[x, y]$.

To do so, the algorithm initially checks which connected components, if any, are large enough at time x, and ignores all the rest. Then, the algorithm treats each and every one of these large components similarly, but separately. Namely, for each one of them the algorithm finds a spanning tree T and sorts all its edges according to the finish time of their availability interval, considering only the interval that includes time x. If the same tree remains connected during $[x, y]$, then the algorithm returns the respective component. Otherwise, if the tree disconnects at a moment t_0 in time, the algorithm employs a similar process to the one used in Algorithm 1, i.e., tries to reconnect the remainder of the tree via edges that are available at t_0. If T cannot be re-connected, then the algorithm checks the sizes of its connected components; it ignores those that are not large enough, while "processing" the rest similarly and separately as before. For each component that is ignored in the process, the algorithm returns the duration of its survival, meaning how long its vertices stayed connected since time x. The algorithm stops when there are no more components that are large enough or when the last component stays connected until time y.

Correctness of the algorithm A component (subgraph) of $G(L)$ is connected during a period $[x, y]$ if and only if there are paths between all pairs of its vertices for all $t \in [x, y]$, i.e., if it induces a spanning tree for every time instance in $[x, y]$. We will now show that if there is a large enough component of $G(L)$ that is connected during $[x, y]$ then Algorithm 2 will find it, and if there is no such component of $G(L)$ then Algorithm 2 will answer that. Clearly, if at time x a connected component of $G(L)$ is not large enough, then we can ignore it; even if at a later moment in time, its vertices connect with more vertices forming a large enough component, that newly formed component is not one that survived throughout $[x, y]$.

Suppose that no large enough connected component of the given $G(L)$ survives through $[x, y]$. Then, either there is no large enough component connected at time x, or there are large enough connected components at time x that get disconnected at some later point $t_0 \in (x, y]$, and break into small connected components. In the first case, Algorithm 2 will delete all the components induced by $E(x)$ and terminate. In the second case, it only processes the large enough components, separately; again, we separate the components because, even if at a later moment in time their vertices connect with other vertices forming a large enough component, that newly formed component is not one that survived throughout $[x, y]$. Since no component survives through $[x, y]$, there is a breaking point for each component, at which there is no tree connecting its vertices. Algorithm 2 realizes the latter (cf. line 20), removes all small sub-components and continues processing the remaining ones separately. Clearly, for each of these sub-components there will be a moment where they disconnect without possibility of re-connection and with all their components being not large enough; that is where the algorithm terminates (cf. line 23).

Now, suppose that the given $G(L)$ has large enough components that remain connected in $[x, y]$. Again, we can view each component separately. Let T be the spanning tree of the component that the algorithm considers at time x. If all the edges of T remain available until time y, then the algorithm will answer that the component remains connected throughout $[x, y]$. If, however, there are edges in T that stop existing at some moment in $[x, y]$, then the algorithm considers the first such moment, t_0. It removes from T all the edges that stop existing at time t_0. T is now disconnected but still contains all the vertices of G. If the subgraph of $G(L)$ induced by the vertices of T and the edges between them available at time t_0 is connected, then the algorithm finds edges to reconnect T (analysis is similar to the analysis of Algorithm 1). If, however, the subgraph is not connected, then the algorithm connects as many components of the subgraph as can be connected (argument similar to the one used in Algorithm 1); it then ignores the small components and proceeds to process the large enough components. Since $G(L)$ has large enough components that remain connected in $[x, y]$, eventually the algorithm finds the component(s) that remain connected throughout $[x, y]$ and terminates (cf. line 8).

Running time The running time of Algorithm 2 depends on the number of times that the spanning trees of the connected components change during $[x, y]$. The spanning trees can only change when one or more of their edges stop being available. The edges on different spanning trees are not overlapping, so the above number is in general upper bounded by the total number of intervals assigned to the edges of the network:

$$M = \sum_{e \in E} |L_e|$$

Initially, to find $E(x)$ we need to look at every edge $e \in E$ and decide if x is between the start and finish time of one of e's availability intervals. Performing a binary search on the ordered set of start times *and* the ordered set of finish times of e's availability intervals, we can decide if $e \in E(x)$ in time $O(\log |L_e|)$. So, to compute $E(x)$, distinguish the connected components and remove those that are not large enough, we need time:

$$M' = O(\sum_{e \in E} \log |L_e|)$$

After that, the algorithm proceeds for each component separately. It is easy to see that the running time of the algorithm for each separate component is the same as the running time of Algorithm 1. Since there are at most $\frac{1}{\varepsilon}$ connected components of size at least $\varepsilon \cdot n$ in $G(L)$ during $[x, y]$, the running time of Algorithm 2 is $O\Big(\frac{1}{\varepsilon}\big(M \cdot (M' + n^2)\big)\Big)$.

5 Low cost maintenance of a tree structure

In this section, we consider an interval temporal network on an underlying clique of n nodes, i.e., all $\binom{n}{2}$ links between nodes of the network *may* exist.

The connectivity of the network needs to be maintained at all moments in time via a tree structure, i.e., a spanning tree of the clique. Each node of the tree performs an individual application determined by the operator of the structure and each link (edge) is active (alive) during a time-interval also decided by the operator, after which the link fails. We have the liberty to provide the operator with extra edges from the clique to re-connect a spanning tree when a link fails; note here that after a new edge is added to the tree structure, the operator then assigns to it a "lifetime" interval, which is determined by the application, anew. The extra edges that we can provide come from the edges of the clique that are not currently used in the tree structure, i.e., a total of $\binom{n}{2} - (n-1)$ edges. We need to assign to every such edge e out of the $\binom{n}{2} - (n-1)$ an availability interval, I_e, so that when the tree structure becomes dis-connected, there is an appropriate such edge available to re-connect it. We call those edges *reserved* edges and the set that consist exactly of all those edges (with their availability intervals) *reservoir*, denoted by R.[6]

Definition 3 (Cost of the reservoir). *The cost of the reservoir is defined as the sum, over all reserved edges, of the length of the edges' availability interval:*

$$c = \sum_{e \in R} |I_e|$$

Let T be the tree structure that is handled by the operator. We consider the time period between 0 and n and we assume that the breaks/failures in the connectivity of T happen once inside every consecutive time interval of length $\Delta \geq \alpha \log n$, for some $\alpha > 1.2$ (*Low-frequency-of-link-breaks assumption*). We are not able to predict when exactly the failures happen, nor are we able to foresee which link will fail next. We also assume worst case breaks in the tree topology within each Δ-interval. The trivial design of the availabilities of the reserved edges would be to make them all available throughout the considered time period $[0, n]$. However, this yields cost $c = \sum_{i=1}^{\binom{n}{2}-n+1} n \in O(n^3)$. We will show how to provide the network with available reserved edges with lower cost, so that the network connectivity is maintained with high probability (whp).[7] In order to re-connect the tree in the worst case of breaks in the tree topology, each reserved edge needs to have been randomly assigned to an availability interval to allow for the same probability of re-connection for all edges.

Theorem 3. *Let $\alpha \in \{x \in \mathbb{R} | x \geq 0.75\}$. If failures of the edges happen once in every consecutive $\Delta \geq \alpha \log n$ time-intervals, then there exists a reservoir of cost $O(n^2 \log n)$ that keeps a spanning tree available during $[0, n]$ whp.*

[6] Notice, here, the distinction between the *availability* of an edge and the *lifetime* of an edge: availability refers to the interval that we assign to a reserved edge with the purpose to re-connect the tree when it breaks, and lifetime refers to the interval that the operator assigns to an edge after it is inserted in the tree structure and is the time interval after which the respective link in the tree structure will fail.

[7] An event occurs with high probability if, for any $\gamma \geq 1$, the event occurs with probability at least $1 - \frac{c_\gamma}{n^\gamma}$, where c_γ depends only on γ.

Proof. Partition the time interval $[0, n]$ into consecutive equisized sub-intervals $b_1, b_2, \ldots, b_{\frac{n}{\beta \log n}}$ of length $\beta \log n,\ \beta \in \mathbb{R},\ 0.75 \leq \beta \leq \alpha$, called *boxes*.[8] For every reserved edge $e \in R$ independently, select a box uniformly at random to be the availability interval of e. For every edge $e \in R$, the probability that e is assigned a particular box $b_i, i = 1, 2, \ldots, \frac{n}{\beta \log n}$ as its availability interval is:

$$Pr[I_e = b_i] = \frac{\beta \log n}{n}$$

Denote by m' the number of edges in R that are assigned to a particular box $b_i,\ i = 1, 2, \ldots, \frac{n}{\beta \log n}$, $m' = |\{e \in R : I_e = b_i\}|$. The expected value of m' is:

$$\mu = E[m'] = \frac{\beta \log n}{n} \cdot \big(\frac{n(n-1)}{2} - n + 1\big) = \frac{\beta n \log n}{2} - \frac{3\beta \log n}{2} + \frac{\beta \log n}{n}$$

By Chernoff bounds, we get that the probability that m' is *close* to the expected number of edges in a particular box $b_i, i = 1, 2, \ldots, \frac{n}{\beta \log n}$ is:

$$\begin{aligned} Pr[m' \in (1 \pm \frac{1}{2})\mu] &\geq 1 - e^{-\frac{1}{4}\mu} \\ &= 1 - e^{-\frac{1}{4} \cdot \frac{\beta \log n}{n} \cdot (\frac{n^2}{2} - \frac{n}{2} - n + 1)} \\ &\geq 1 - \frac{1}{n^{\frac{\beta n}{16}}}, \text{ for } n \text{ large enough } (n \geq 6) \end{aligned}$$

We now show that when a failure happens in T, we can whp find an edge in R which is available at that particular moment in time. Consider the specific box b_i that includes the time moment at which the failure in T happens. The number of edges in b_i that can re-connect T depends on where the failure happens, i.e., on the sizes[9] of the two connected components after the failure. If n_1 and n_2 are the sizes of the connected components of T after a failure, then the probability that a particular edge $e \in b_i$ can re-connect T after being added to the structure is:

$$Pr[e \in b_i \text{ re-connects } T] = \frac{n_1 \cdot n_2}{\frac{(n_1+n_2)\cdot(n_1+n_2-1)}{2}} \geq \frac{2n_1n_2}{(n_1+n_2)^2}$$

The probability that no edge of b_i reconnects T after a failure is:

$$Pr[\text{no } e \in b_i \text{ re-connects } T] \leq \big(1 - \frac{2n_1n_2}{(n_1+n_2)^2}\big)^{m'}$$

So, the probability that there is an edge in b_i that re-connects T is:

$$Pr[b_i \text{ re-connects } T] \geq 1 - \big(1 - \frac{2n_1n_2}{(n_1+n_2)^2}\big)^{m'} \geq 1 - \big(1 - \frac{2n_1n_2}{(n_1+n_2)^2}\big)^{\frac{3\mu}{2}}$$

[8] The last box is not necessarily of size exactly $\beta \log n$ but this does not affect the analysis.

[9] The size of a component is the number of its vertices.

In the worst case, T dis-connects into a component of size $n-1$ and a single vertex. So, we can reconnect T after a failure with probability:

$$\begin{aligned} Pr[b_i \text{ re-connects } T] &\geq 1-\big(1-\frac{2(n-1)}{n^2}\big)^{\frac{3\beta n\log n}{4}-\frac{9\beta\log n}{4}+\frac{3\beta\log n}{2n}} \\ &\geq 1-\big(1-\frac{2(n-1)}{n^2)}\big)^{\frac{3}{4}\beta n\log n} \\ &\geq 1-\frac{1}{n^{0.9\frac{3}{2}\beta}}, \text{ for } n \text{ large enough } (n\geq 10) \\ &= 1-\frac{1}{n^{1.35\beta}} \xrightarrow{n\to+\infty} 1 \end{aligned}$$

We require $\beta \geq 0.75$ so that the above event happens whp. The probability that within the time period $[0,n]$, there is a box that will not re-connect T is:

$$\begin{aligned} Pr[\exists b_i,\ i=1,\ldots,\frac{n}{\beta\log n} : b_i \text{ doesn't re-connect } T] &\leq \sum_{i=1}^{\frac{n}{\beta\log n}} \frac{1}{n^{1.35\beta}} \\ &= \frac{n}{\beta\log n}\cdot\frac{1}{n^{1.35\beta}} \xrightarrow{n\to+\infty} 0 \end{aligned}$$

So, we can almost surely[10] re-connect T during $[0,n]$ by employing the above random assignment of availability intervals to the reserved edges, having total cost $c=\sum_{i=1}^{\binom{n}{2}-n+1}\beta\log n \in O(n^2\log n)$. □

Conjecture 1. If failures of the edges happen once in every consecutive $\Delta \geq \alpha\log n$ time-intervals, we conjecture that there is *no* reservoir of cost $o(n^2\log n)$ that keeps a spanning tree available during $[0,n]$ whp.

Open Problem 1 *For spanning tree breaks of frequency $o(\log n)$ within the time period $[0,n]$, the reservoir of Theorem 3 does not re-connect T whp. It remains an open question to derive a scheme that does so for breaks of so high frequency.*

Open Problem 2 *What is a low cost reservoir to maintain a spanning tree of the clique network, if the failures in the links of the tree happen randomly, e.g., if each link receives a lifetime given by the Exponential Distribution?*

6 Conclusions and further research

In this work, we study the verification and maintenance of the connectivity in interval temporal networks. We pose an open question to maintain the spanning tree in the interval $[0,n]$ with cost $o(n^2\log n)$. An extension of this research includes the construction of a reservoir to maintain a spanning tree of the clique network, in cases where the failures in the links of the tree happen randomly, e.g., if each link receives a lifetime given by the Exponential Distribution.

[10] Note that increasing the size of the boxes by a constant factor, i.e., increasing the lower bound for β and α, can enforce the re-connection probability to also increase.

References

1. E. C. Akrida, L. Gąsieniec, G. B. Mertzios, and P. G. Spirakis. Ephemeral networks with random availability of links: Diameter and connectivity. In *SPAA*, 2014.
2. D. Angluin, J. Aspnes, Z. Diamadi, M. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, pages 235–253, 2006.
3. C. Avin, M. Koucký, and Z. Lotker. How to explore a fast-changing world (cover time of a simple random walk on evolving graphs). In *ICALP*, pages 121–132, 2008.
4. B.-M. Bui-Xuan, A. Ferreira, and A. Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(2):267–285, 2003.
5. A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems (IJPEDS)*, 27(5):387–408, 2012.
6. A. E. F. Clementi, C. Macci, A. Monti, F. Pasquale, and R. Silvestri. Flooding time of edge-markovian evolving graphs. *SIAM Journal on Discrete Mathematics*, 24(4):1694–1712, 2010.
7. C. Cooper, R. Klasing, and T. Radzik. A randomized algorithm for the joining protocol in dynamic distributed networks. *Theoretical Computer Science*, 406(3):248–262, 2008.
8. P. Duchon and R. Duvignau. Local update algorithms for random graphs. In *LATIN*, pages 367–378, 2014.
9. C. Dutta, G. Pandurangan, R. Rajaraman, Z. Sun, and E. Viola. On the complexity of information spreading in dynamic networks. In *SODA*, pages 717–736, 2013.
10. L. Fleischer and M. Skutella. Quickest flows over time. *SIAM Journal of Computing*, 36(6):1600–1630, 2007.
11. L. Fleischer and É. Tardos. Efficient continuous-time dynamic network flow algorithms. *Operations Research Letters*, 23(3-5):71–80, 1998.
12. C. Gavoille, D. Peleg, S. Perennes, and R. Raz. Distance labeling in graphs. In *SODA*, pages 210–219, 2001.
13. M. Katz, N. A. Katz, A. Korman, and D. Peleg. Labeling schemes for flow and connectivity. *SIAM Journal on Computing*, 34(1):23–40, 2004.
14. D. Kempe, J. M. Kleinberg, and A. Kumar. Connectivity and inference problems for temporal networks. In *STOC*, pages 504–513, 2000.
15. R. Koch, E. Nasrabadi, and M. Skutella. Continuous and discrete flows over time - A general model based on measure theory. *Mathematical Methods of Operations Research*, 73(3):301–337, 2011.
16. F. Kuhn, N. A. Lynch, and R. Oshman. Distributed computation in dynamic networks. In *STOC*, pages 513–522, 2010.
17. G. B. Mertzios, O. Michail, I. Chatzigiannakis, and P. G. Spirakis. Temporal network optimization subject to connectivity constraints. In *ICALP*, pages 657–668, 2013.
18. O. Michail, I. Chatzigiannakis, and P. G. Spirakis. Causality, influence, and computation in possibly disconnected synchronous dynamic networks. In *OPODIS*, pages 269–283, 2012.
19. M. Molloy and B. Reed. *Graph colouring and the probabilistic method*, volume 23 of *Algorithms and Combinatorics*. Springer, 2002.

20. R. O'Dell and R. Wattenhofer. Information dissemination in highly dynamic graphs. In *DIALM-POMC*, pages 104–110, 2005.
21. C. Scheideler. Models and techniques for communication in dynamic networks. In *STACS*, volume 2285, pages 27–49, 2002.